La vista: ActionForms y taglibs propias

Índice

En este tema trataremos sobre la vista en Struts. En este aspecto el framework nos aporta unos componentes muy útiles que actúan de interfaz entre la vista y las acciones: los *ActionForms*. Como veremos, estos objetos nos permiten recoger automáticamente los valores de los formularios y validar sus datos, entre otras funciones. Veremos también las librerías de etiquetas propias de Struts. Aunque algunas de ellas han quedado obsoletas tras la aparición de JSTL, otras siguen siendo de utilidad en conjunto con los ActionForms.

1. ActionForms

1.1. Introducción

Aunque los datos introducidos en formularios pueden obtenerse dentro del código de las acciones directamente de la petición HTTP (como hemos visto en los ejemplos del tema anterior), Struts ofrece un mecanismo alternativo que proporciona distintas ventajas: los **ActionForm**s. Empleando ActionForms podemos conseguir:

- **Recolección automática de datos** a partir de los incluidos en la petición HTTP.
- **Validación de datos modular** (realizada fuera del código de la acción), y en caso de utilizar el plugin *Validator*, validación automática según lo especificado en un fichero de configuración.
- **Recuperación de los datos** para volver a rellenar formularios. De esta forma se evita el típico problema de que el usuario tenga que volver a rellenar un formulario entero porque uno de los datos es incorrecto.

Podemos considerar un ActionForm como si fuera un JavaBean que captura los datos de un formulario. Los datos se pueden extraer, validar, cambiar y volver a colocar en otro formulario. No obstante, no tiene por qué haber una correspondencia uno a uno entre un ActionForm y un formulario HTML, de manera que se puede utilizar el mismo ActionForm para englobar varios formularios separados en distintas páginas (caso típico de un asistente). También se puede reutilizar el mismo ActionForm en distintos formularios que compartan datos (por ejemplo, para dar de alta o modificar los datos de un usuario registrado).

En Struts hay dos tipos principales de ActionForms

- Clases descendientes de la clase base **ActionForm**. Deben incorporar métodos Java para obtener/cambiar cada uno de los datos (al estilo JavaBeans), cuya definición puede resultar tediosa. Desde Struts 1.1 las propiedades se pueden almacenar en un Map, con lo que solo es necesario definir un único método para acceso a todas las propiedades, pasando como parámetro el nombre de la deseada.
- Instancias de la clase **DynaActionForm** o descendientes de ésta. Permiten definir los campos en el fichero `struts-config.xml`, de manera que se pueden añadir o modificar campos minimizando la necesidad de recompilar código. Mediante la clase

DynaValidatorForm se puede incluso validar datos de manera automática, según las reglas especificadas en un fichero de configuración aparte.

1.2. El ciclo de vida de un ActionForm

La generación y procesamiento de un ActionForm pasa por varias etapas:

1. El controlador recibe la petición del usuario, y chequea si la acción asociada utiliza un ActionForm. De ser así, crea el objeto
2. Se llama al método `reset()` del objeto, que el desarrollador puede sobreescribir para "limpiar" sus campos. Esto tiene sentido si el ActionForm persiste más allá de la petición actual (lo cual se puede especificar al definirlo).
3. El ActionForm se almacena en el ámbito especificado en su definición (petición, sesión o aplicación)
4. Los datos del ActionForm se rellenan con los que contiene la petición HTTP. Cada parámetro HTTP se asocia con el dato del mismo nombre del ActionForm. Un punto importante es que en HTTP los parámetros son cadenas, con lo que en principio las propiedades del ActionForm deben ser Strings, aunque los valores booleanos se convierten a boolean. No obstante, más allá de esta conversión básica no hay conversión automática de tipos.
5. Se validan los datos, llamando al método `validate()`, que el desarrollador debe sobreescribir para implementar la validación deseada.
6. Si se han producido errores de validación, se efectúa una redirección a la página especificada para este caso. Si no, se llama al `execute()` de la acción y finalmente se muestra la vista asociada a esta.
7. Si en la vista se utilizan las *taglibs* de Struts para mostrar datos, aparecerán los datos del ActionForm.

1.3. Cómo definir un ActionForm

Los ActionForm se definen dentro del fichero `struts-config.xml`, dentro de la sección `<form-beans>`. Cada ActionForm viene definido por un elemento `<form-bean>`. Por ejemplo, para definir un ActionForm con el nombre `FormLogin` y asociado a la clase Java `acciones.forms.FormLogin` haríamos:

```
<form-beans>
   <form-bean name="FormLogin" type="acciones.forms.FormLogin">
</form-beans>
```

Dentro de un momento veremos qué condiciones debe cumplir la clase `acciones.forms.FormLogin`, que tendremos que definir nosotros.

Para que los datos de un `ActionForm` sean accesibles a una acción, hay que definir una serie de atributos dentro del elemento `action` de `struts-config.xml`. Por ejemplo, para asociar un ActionForm de la clase `FormLogin` a la acción `login` de los ejemplos

anteriores, se podría hacer:

```
<action path="/login" type="acciones.AccionLogin"
        name="FormLogin" scope="session"
        validate="true" input="/index.jsp">
   <forward name="OK" path="/personal.jsp"/>
   <forward name="errorUsuario" path="/error.html"/>
</action>
```

El atributo `name` indica el nombre simbólico para el `ActionForm`. El `scope` tiene el mismo significado que cuando tratamos con JavaBeans. En caso de que se desee validar los datos, hay que especificar el atributo `validate=true` y utilizar el atributo `input` para designar la página a la que se volverá si se han producido errores de validación.

Nota:

Como puede verse, los nombres de los atributos del ActionForm en el XML no son demasiado descriptivos del papel que desempeñan (excepto, quizás, scope). Este es un problema reconocido por los propios desarrolladores de Struts y que se soluciona en la versión 2 del framework.

Por otro lado hay que escribir el código Java con la clase del ActionForm. Esta tarea es muy similar a definir un JavaBean. Hay que especificar métodos get/set para cada campo, y colocar la lógica de validación en el método `validate()`. Por ejemplo:

```
package acciones.formularios;
import org.apache.struts.action.*;
import javax.servlet.http.HttpServletRequest;

public class FormLogin extends ActionForm {
   private String login;
   private String password;

   public void setLogin(String login) {
      this.login = login;
   }
   public String getLogin() {
      return login;
   }

   public ActionErrors validate(ActionMapping mapping,
                                HttpServletRequest request) {
      ActionErrors errores = new ActionErrors();
      if ((getlogin()==null)||(getLogin.equals("")))
         errores.add(ActionMessages.GLOBAL_MESSAGE
                     , new
ActionMessage("error.requerido.usuario"));
      return errores;
   }
}
```

Nótese que el ActionForm no es más que un bean, eso sí, debe heredar de la clase `ActionForm`.

El método `validate()` debe devolver una colección de errores (o null), representada por el objeto **ActionErrors**.

Aviso:
Si en el struts-config.xml ponemos validate="true" o no especificamos el valor de este atributo, Struts llamará al método validate() de nuestro ActionForm, generando un error si éste no existe. Si no deseamos validar los datos, debemos ponerlo específicamente con validate="false"

Desde la versión 1.1 de Struts, se pueden utilizar ActionForms que almacenen internamente las propiedades en un objeto Map (*Map-backed Action Forms)*. Para el acceso a/modificación de cualquier propiedad solo será necesario implementar dos métodos

```
public void setValue(String clave, Object valor)
public Object getValue(String clave)
```

1.4. Tipos de datos del ActionForm: conversión y validación

En el ejemplo anterior, todas las propiedades del ActionForm eran Strings, pero esto no tiene por qué ser siempre así. Una propiedad de un ActionForm puede ser de cualquier tipo, pero como Struts copia automáticamente los valores de la petición HTTP a las propiedades del ActionForm tiene que "saber hacer" la conversión de tipos. Struts puede convertir de String (el tipo que tienen todos los parámetros HTTP) a cualquier tipo primitivo de Java. El problema es que si usamos una propiedad de tipo int, por ejemplo, y el usuario escribe en el campo de formulario correspondiente algo que no se puede convertir a entero, la conversión fallará y el campo pasará a valer 0. Esto puede causar problemas si queremos mostrar de nuevo los valores introducidos para que el usuario los pueda corregir (veremos cómo se hace esto con las taglibs de Struts). Para el usuario puede ser un poco desconcertante ver un 0 donde antes se había tecleado otra cosa totalmente distinta. Adoptando por tanto un punto de vista pragmático, muchos desarrolladores de Struts usan solo Strings para las propiedades de los ActionForms y se ocupan manualmente de hacer la conversión y de detectar los posibles errores. Esto es tedioso, pero es difícil de evitar dado el diseño de Struts.

1.5. DynaActionForms

En ActionForms con muchos campos, resulta tedioso tener que definir manualmente métodos *get/set* para cada uno de ellos. En su lugar, podemos utilizar `DynaActionForms`, en los que los campos se definen en el fichero de configuración de Struts. Por ejemplo:

```
 <form-bean name="FormLogin"
type="org.apache.struts.action.DynaActionForm">
   <form-property name="login" type="java.lang.String"/>
```

```
  <form-property name="password" type="java.lang.String"/>
 </form-bean>
```

Como se ve, en este caso la clase del ActionForm no la definimos nosotros sino que es propia de Struts. Para acceder a los campos del `DynaActionForm` se usa un método genérico **get** al que se pasa como parámetro el nombre del campo (por ejemplo `get("login")`). Algo similar ocurre para cambiar el valor de un campo (método **set**).

Nota:

En JSP, el acceso a una propiedad de un DynaActionForm con EL se debe hacer a través de la propiedad predefinida `map`. Así en el ejemplo anterior, la propiedad login sería accesible con ${*nombreDelBean*.map.login}

Para validar un `DynaActionForm`, tendremos que definir una clase propia que herede de `org.apache.struts.action.DynaActionForm`, y sobreescribir su método `validate()`, o mejor aún, podemos validar datos automáticamente utilizando el plugin de Struts denominado **validator**, que será objeto del siguiente tema.

2. La taglib HTML de Struts

Struts viene con varias taglibs: de ellas, la más importante es la HTML, que es la que sirve para capturar/mostrar los datos de los ActionForms. Esta taglib es muy útil en cualquier aplicación de Struts. Entre otras cosas, podemos volver a mostrar los valores introducidos en un formulario si se ha producido un error en alguno sin que el usuario tenga que rellenarlos todos de nuevo y generar *checkboxes*, opciones de un `select` etc a partir de colecciones o arrays, sin tener que hacerlas una por una.

Las otras taglibs (*logic*, *bean*,...) aunque también muy útiles en su momento, quedaron obsoletas con la aparición de JSTL. Por tanto discutiremos aquí únicamente la taglib de HTML. Además, nos centraremos en la definición de formularios y campos de formulario, aunque la taglib tiene otras etiquetas adicionales.

Para poder usar la taglib HTML en un JSP, hay que incluir en el JSP la directiva

```
  <%@ taglib uri="http://struts.apache.org/tags-html" prefix="html"
%>
```

La versión "clásica" de la taglib HTML no permite el uso del lenguaje de expresiones (EL). Si deseamos usarlo en nuestras páginas, hay que importar a nuestro proyecto el archivo `struts-el.jar` y usar en los JSPs la directiva

```
 <%@ taglib uri="http://struts.apache.org/tags-html-el"
prefix="html-el" %>
```

Aviso:

En algunas versiones anteriores de Struts las URIs comienzan por jakarta.apache.org en lugar de struts.apache.org, consecuencia del cambio de dominio del proyecto Struts.

2.1. Definir el formulario

Lo primero que necesitamos para definir un formulario con las taglibs de Struts es una acción a ejecutar y un ActionForm asociado a dicha acción. Sin acción o ActionForm no tiene mucho sentido usar para el formulario la taglib de Struts (usaríamos directamente un formulario HTML).

En general, las etiquetas de la taglib HTML de Struts son muy similares a su contrapartida HTML. El caso de la definición de formulario no es una excepción. Por ejemplo, vamos a ir viendo cómo se podría hacer un formulario de registro de nuevo usuario con las taglibs de Struts:

```
  <%@ taglib uri="http://struts.apache.org/tags-html" prefix="html"
%>
  <html>
  <head><title>Ejemplo de tag form<title></head>
  <body>
    <html:form action="/registrar">
     <-- aquí vienen los campos. Ahora veremos cómo se definen -->
    </html:form>
  </body>
  </html>
```

La primera diferencia que se puede observar entre este formulario y uno HTML (aparte del prefijo `html:`) es que el nombre del `action` no es una URL "física" sino una acción de Struts.

El primer campo que necesitamos para que al menos los datos lleguen al servidor es un botón de tipo `submit`. Struts nos ofrece además la posibilidad de definir un botón para cancelar. Veamos de nuevo el ejemplo anterior (pero ahora omitiendo el HTML de fuera del formulario).

```
<html:form action="/registrar">
   <html:submit>Registrar nuevo usuario</html:submit>
   <html:cancel>Cancelar</html:cancel>
</html:form>
```

Como vemos, la principal diferencia de sintaxis con el HTML "puro" es que el "cartel" del botón se pone dentro de la etiqueta, en lugar de con el atributo "value". La acción asociada se disparará se haya pulsado en el submit o en el cancel, pero en este último caso

el método `isCancelled` de la clase `Action` devolverá `true`.

```
...
 public ActionForward execute(
         ActionMapping mapping,
         ActionForm form,
         HttpServletRequest request,
         HttpServletResponse response)
         throws Exception {
         ...
         if (isCancelled(request)) {
                 ...
         }
         ...
}
```

Cuando una acción se cancela no se efectúa la validación del ActionForm. Struts controla la cancelación enviando un parámetro HTTP especial, lo cual puede dar lugar a que usuarios malintencionados intenten manipular este parámetro para evitar la validación. En Struts 1.2.X, a partir de la versión 1.2.9 cuando una acción es cancelable hay que especificarlo en el `struts-config.xml`:

```
<action path="/registrar"
        input="/registrar.jsp"
        validate="true">
    <set-property property="cancellable" value="true"/>
    <forward name="OK" path="/resultado.jsp"/>
</action>
```

En caso de no hacer esto, la acción lanzará una excepción de tipo `InvalidCancelException`.

A partir de Struts 1.3.X esto se hace con un atributo `cancellable`

```
<action path="/registrar"
        input="/registrar.jsp"
        validate="true" cancellable="true">
    <forward name="OK" path="/resultado.jsp"/>
</action>
```

2.2. Campos de texto

Los campos para introducir texto son muy similares a los de HTML, con la diferencia de que en lugar de elegir el tipo de campo con el atributo type se hace con la propia etiqueta. El atributo property nos permite asociar un campo a la propiedad del mismo nombre del ActionForm asociado al formulario. Por ejemplo:

```
<html:form action="/registrar">
   <html:text property="login">Introduce aquí el nombre que deseas
tener</html:text>
   <html:password property="password"/>
   <html:submit>Registrar nuevo usuario</html:submit>
   <html:cancel>Cancelar</html:cancel>
</html:form>
```

Para que el ejemplo funcione, debemos tener un ActionForm asociado a la acción "registrar" que tenga al menos las propiedades login y password. Como ya se ha visto en el ciclo de vida de los ActionForms, cuando se rellena el formulario Struts copiará los datos introducidos en las propiedades correspondientes del ActionForm. A la inversa, si el método validate() del ActionForm devuelve false y el struts-config.xml indica que hay que volver a la misma página, Struts copiará los datos desde el ActionForm al formulario, para que el usuario pueda corregirlos.

En el caso de las contraseñas, puede que no resulte adecuado volver a mostrar el valor introducido. Para ello se utiliza el atributo `redisplay`, al que hay que asignar el valor `false`

Como es de esperar a estas alturas (visto el resto de etiquetas) los campos de texto multilínea se consiguen con la etiqueta `html:textarea`.

```
<html:textarea property="descripcion" cols="40" rows="6"/>
```

2.3. Cuadros de lista

La sintaxis de los cuadros de lista es prácticamente idéntica a la de HTML, si es que vamos a poner las opciones manualmente, una por una:

```
<html:select property="sexo">
   <html:option value="H">Hombre</html:option>
   <html:option value="M">Mujer</html:option>
   <html:option value="N">No especificado</html:option>
</html:select>
```

Struts asociará este campo a la propiedad "sexo" del ActionForm correspondiente a la acción ejecutada

2.3.1. Tomar los valores para "option" automáticamente de un objeto

Lo interesante es que Struts puede tomar los valores del select de una colección o un array, ahorrándonos el tener que especificar las opciones "a mano". Para ello, debemos

tener en la petición o la sesión el array con las opciones. La manera más sencilla de conseguir esto (sin meterlo en el propio JSP) es ponerlo en la acción que nos lleva a la página con el formulario. Veamos un ejemplo, con el siguiente struts-config.xml

```
...
<form-beans>
   <form-bean name="FormRegistro" type="actionForms.FormRegistro"/>
</form-beans>
...
<action-mappings>
    <action
       path="/prepararRegistro"
       type="acciones.AccionPrepararRegistro">
       <forward name="OK" path="/registro.jsp"/>
    </action>
    <action
       path="/registrar"
       name="FormRegistro" scope="request"
       type="acciones.AccionRegistrar">
       <forward name="OK" path="/resultRegistro.jsp"/>
    </action>
</action-mappings>
...
```

Como vemos, la acción `AccionPrepararRegistro` muestra la página `registro.jsp`, que contiene el formulario que veíamos en los ejemplos anteriores. Este formulario llama a `registrar.do`, que resulta en la ejecución de `AccionRegistrar`. El sitio más adecuado para colocar en algún ámbito (petición, sesión o aplicación) la lista de opciones para el select sería entonces en `AccionPrepararRegiatro`, simplemente haciendo algo como

```
String[] lista = { "Hombre", "Mujer", "No especificado"};
request.setAttribute("listaSexos", lista);
```

Nótese que aunque hemos usado un array, `listaSexos` podría ser cualquier tipo de `Collection`

Ahora podemos decirle a Struts en el HTML que tome las opciones del objeto `listaSexos`. Automáticamente buscará este en la petición, sesión y aplicación, por este orden.

```
<html:select property="sexo">
   <html:options value="listaSexos"/>
</html:select>
```

Esto nos generará un select en HTML en el que tanto los atributos "value" como las etiquetas que se muestran en pantalla se toman de `listaSexos`. Si deseamos que los value sean distintos a las etiquetas haríamos algo como

```
<html:select property="sexo">
   <html:options name="codSexos" labelName="listaSexos"/>
</html:select>
```

Así, los valores del atributo value se tomarán del objeto `codSexos` (que debemos haber instanciado antes) y las etiquetas que se muestran en pantalla de `listaPerfiles`

2.3.2. Tomar los valores para "option" automáticamente de un ActionForm

Podemos también tomar los valores para las etiquetas "option" de un ActionForm. Lo más lógico es usar el asociado al formulario (hablando con más propiedad, el asociado a la acción disparada por el formulario). En ese caso, debemos añadir a nuestro ActionForm un método "getXXX" que devuelva un array o colección con el que se rellenará el cuadro de lista. Por ejemplo:

```
  public class FormRegistro {
   ...
   private static String[] listaSexos = { "Hombre", "Mujer", "No
especificado"};

   public String[] getListaSexos() {
           return listaSexos;
   }
   ...
```

Ahora, en el JSP hacemos

```
...
<html:select property="sexo">
  <html:options property="listaSexos"/>
</html:select>
...
```

También podemos tomar de un "getXXX" los valores del atributo "value" y de otro "getYYY" las etiquetas que se muestran en pantalla. Para especificar de dónde sacar las etiquetas, usaremos el atributo `labelProperty` de `<html:options>`.

2.4. Botones de radio

La etiqueta de Struts para definir botones de radio es prácticamente equivalente a la de HTM, con la diferencia básica de que podemos asociar el botón a una propiedad de un ActionForm:

```
<html:radio property="sexo" value="hombre"/> hombre
<html:radio property="sexo" value="mujer"/> mujer
```

Para generar varios botones de radio con distintos `value` de manera automática no tendremos más remedio que usar JSTL, ya que Struts no ofrece ningún mecanismo para hacerlo. Por ejemplo, supongamos que el ActionForm `FormRegistro` tiene un método `getListaSexos` que devuelve los valores posibles para el sexo, como en el ejemplo de la sección anterior. Podríamos hacer:

```
<%@ taglib uri="http://struts.apache.org/tags-html-el"
prefix="html-el" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
...
<c:forEach items="${FormRegistro.listaSexos}" var="sexo">
   <html-el:radio property="sexo" value="${sexo}"/> ${sexo}
</c:forEach>
```

Nótese que en el ejemplo anterior hemos usado la versión EL de la taglib para poder emplear el lenguaje de expresiones dentro de las etiquetas de Struts.

2.5. Casillas de verificación

Struts nos ofrece dos etiquetas para hacer casillas de verificación (*checkboxes*), según si queremos una sola casilla o una serie de casillas relacionadas. En el primer caso, la asociación se hace con una propiedad booleana de un ActionForm y en el segundo con un array de valores, normalmente Strings (ya que en este caso podemos marcar varias casillas simultáneamente)

Cuando queremos una única casilla podemos usar la etiqueta `checkbox`, que es prácticamente equivalente a su contrapartida HTML:

```
<html:checkbox property="publi"/> Sí, deseo recibir publicidad
sobre sus productos
```

El checkbox se asociaría a una propiedad booleana del ActionForm llamada "publi". Deben existir los métodos isPubli/getPubli para que Struts pueda consultar/fijar su valor.

Aviso:

Hay que llevar mucho cuidado con las casillas de verificación si se usa un ActionForm con ámbito de sesión o de aplicación. Cuando el checkbox no se marca, el navegador **no envía** el valor asociado, ni siquiera como false ni nada similar. Esto puede causar el que aunque el usuario ha desmarcado la casilla, la propiedad asociada sigue con true, ya que Struts no ha llamado al setXXX correspondiente. Para evitarlo, se debe inicializar la propiedad asociada a `false`. De este modo, antes de mostrar el formulario Struts pone el valor a `false`. Cuando se envían los datos, si la casilla está marcada cambiará a `true` y si no, no ocurrirá nada. en el método `reset()` del ActionForm

Para hacer más de una casilla de verificación asociada a una sola propiedad de un ActionForm se usa la etiqueta `multibox`. Dentro de la etiqueta hay que poner el valor que se añadirá a la propiedad. Decimos que se añadirá porque la propiedad debe ser un array de valores, donde se colocarán los asociados a las casillas marcadas.

```
<html:multibox property="aficiones">cine</html:multibox> Cine
<html:multibox property="aficiones">música</html:multibox> Música
<html:multibox property="aficiones">videojuegos</html:multibox>
Videojuegos
```

Como en el caso de los botones de radio, si tenemos muchas opciones es más conveniente generar las casillas de manera automática. Esto tendremos que hacerlo iterando sobre una colección con los valores, como en la sección anterior. Suponiendo que el ActionForm tiene un método `getListaAficiones` que devuelve un array con "cine", "música" y "videojuegos", El ejemplo anterior se convertiría en:

```
<c:forEach items="${FormRegistro.listaAficiones}" var="aficion">
   <html-el:multibox
property="aficiones">${aficion}</html-el:multibox> ${aficion}
</c:forEach>
```